LABORATORIO 7

ESTACIÓN METEOROLÓGICA

JOSÉ PRINCE Y DIEGO LINARES

CONEXIÓN AL BROKER:

Servidor: iot.redesuvg.cloud

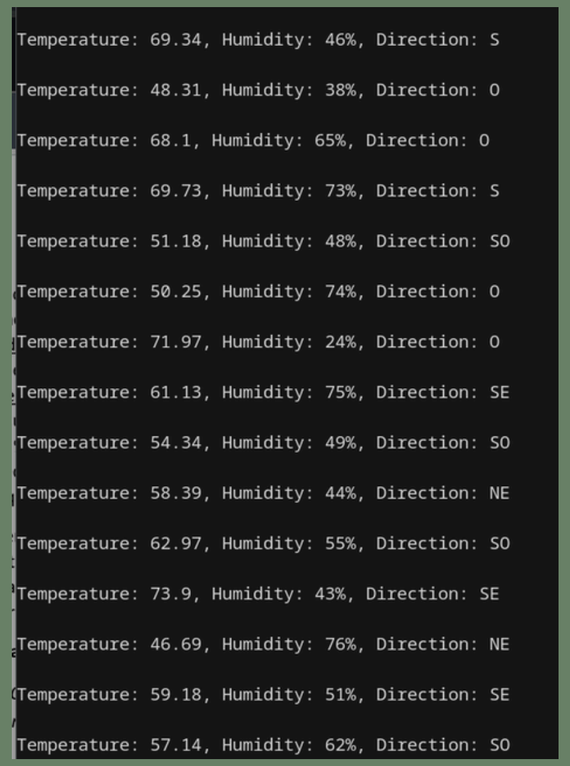
IP: 147.182.219.133

Puerto: 9092

Topic: 22087

3.1 SIMULACIÓN DE UN SENSOR

Datos generados:



- ¿A qué capa pertenece JSON/SOAP según el Modelo OSI y porque?

JSON y SOAP pertenecen a la Capa 6 (Capa de Presentación del modelo OSI). Esta capa se encarga de la representación de los datos y la estructura del mensaje. Lo que hacen tanto JSON como SOAP es definir un formato para estructurar los datos.

- ¿Qué beneficios tiene utilizar un formato como JSON/SOAP?

Beneficios de JSON:

- Liviano y eficiente
- Fácil de leer y entender
- Ideal para microservicios, APIs REST y sistemas Io

Beneficios de SOAP:

- Incluye normas de seguridad complejas.
- Soporta transacciones distribuidas

3.2 ENVÍO DE DATOS AL SERVER EDGE

Implementación de Producer:

```python
from kafka import KafkaProducer
import numpy as np
import time

class Producer:
    def __init__(self, host, port, topic):
        self.host = host
        self.port = port
        self.topic = topic

        self.producer = KafkaProducer(
            bootstrap_servers=[f"{host}:{port}"],
            value_serializer=lambda v: v
        )

    def generate_data(self):
        directions = ["N", "NO", "O", "SO", "S", "SE", "E", "NE"]

        # Temperature
        temp = np.random.normal(loc=55.0, scale=15.0)
        temp = np.clip(temp, 0.0, 110.0) # 0 - 110.00
        temp = round(temp, 2)

        # Humidity
        humidity = np.random.normal(loc=50.0, scale=20.0)
        humidity = int(np.clip(humidity, 0.0, 100.0)) # 0 - 100

        # Direction
        idx = np.random.normal(loc=len(directions)/2, scale=2.0)
        idx = int(np.clip(round(idx), 0, len(directions) - 1))
        direction = directions[idx]

        return float(temp), humidity, direction, idx

    def encode_to_bytes(self, temp, hum, dir_idx):
        # Byte 1
        temp_byte = int((temp / 110.0) * 255) & 0xFF

        # Byte 2
        hum_byte = int((hum / 100.0) * 255) & 0xFF

        # Byte 3
        dir_byte = dir_idx & 0xFF

        return bytes([temp_byte, hum_byte, dir_byte])

    def send_data(self):
        temp, humd, direc, dir_idx = self.generate_data()

        payload = self.encode_to_bytes(temp, humd, dir_idx)

        self.producer.send(self.topic, payload)
        print(f'\nTemperature: {temp}, Humidity: {humd}%, Direction: {direc}')
        self.producer.flush()

if __name__ == "__main__":
    producer = Producer("iot.redesuvg.cloud", 9092, "22087")

    while(True):
        producer.send_data()
        time.sleep(1)
```

- generate_data: genera en distribución normal los datos de temperatura, humedad y dirección del viento.
- encode_to_bytes: encodea los datos para que solo sean 3 bytes los que se envían.
- send_data: envia los datos genrados al broker.

3.3 CONSUMIR Y DESPLEGAR DATOS METEOROLÓGICOS

Implementación de Consumer:

```python
class Consumer:
    def __init__(self, host, port, topic, group_id="weather_station_consumer"):
        self.host = host
        self.port = port
        self.topic = topic

        self.consumer = KafkaConsumer(
            topic,
            bootstrap_servers=[f"{host}:{port}"],
            group_id=group_id,
            auto_offset_reset='latest',
            enable_auto_commit=True,
            value_deserializer=lambda v: v
        )

        # Direcciones del viento (mismo orden que el producer)
        self.directions = ["N", "NO", "O", "SO", "S", "SE", "E", "NE"]

        self.max_points = 50
        self.temperatures = deque(maxlen=self.max_points)
        self.humidities = deque(maxlen=self.max_points)
        self.wind_directions = deque(maxlen=self.max_points)
        self.timestamps = deque(maxlen=self.max_points)

        self.message_count = 0

    def decode_from_bytes(self, payload):
        """
        Decodifica los 3 bytes recibidos a valores originales.

        Byte 1: Temperatura (0-255 -> 0-110.00°C)
        Byte 2: Humedad (0-255 -> 0-100%)
        Byte 3: Índice de dirección del viento (0-7)
        """
        if len(payload) != 3:
            raise ValueError(f"Payload debe ser de 3 bytes, recibido: {len(payload)}")

        # Decodificar temperatura
        temp_byte = payload[0]
        temp = (temp_byte / 255.0) * 110.0
        temp = round(temp, 2)

        # Decodificar humedad
        hum_byte = payload[1]
        humidity = int((hum_byte / 255.0) * 100.0)

        # Decodificar dirección del viento
        dir_idx = payload[2]
        if dir_idx >= len(self.directions):
            dir_idx = 0
        direction = self.directions[dir_idx]

        return temp, humidity, direction
```

- decode_from_bytes: Decodifica los 3 bytes recibidos del producer. Convierte el primer byte a temperatura, el segundo byte a humedad, y el tercer byte al índice de dirección del viento. Retorna los tres valores decodificados.

- setup_plots: Configura la interfaz gráfica con los 4 paneles para temperatura, humedad, frecuencia para direccion de viento y panel de texto para estadisticas.

- update_plots: Consulta Kafka por nuevos mensajes, los decodifica, actualiza las colas de datos, y refresca todos los gráfico
- start_monitoring: Configura los gráficos y arranca la animación que actualiza la visualización cada 500ms mientras escucha mensajes del broker Kafka.
- consume_simple: Imprime los datos recibidos en consola de forma continua. Útil para pruebas rápidas o cuando no se necesita visualización gráfica.

3.4 IOT EN ENTORNOS CON RESTRICCIONES

Encriptación del mensaje:

BYTE de Temperatura:

1. Tomar la temperatura real.
2. Dividirla entre 110 para normalizarla al rango 0-1.
3. Multiplicar ese valor normalizado por 255 para llevarlo al rango 0-255.
4. Convertir el resultado a número entero

BYTE de Humedad:

1. Tomar la humedad real.
2. Dividirla entre 100 para normalizarla al rango 0-1.
3. Multiplicar ese valor normalizado por 255 para llevarlo al rango 0-255.
4. Convertir el resultado a número entero

BYTE de Dirección:

1. Obtener el índice de dirección.
2. Asegurar que el valor está en el rango 0-255.

3.4.1 IOT EN ENTORNOS CON RESTRICCIONES

- ¿A qué capa pertenece JSON/SOAP según el Modelo OSI y por qué?

JSON y SOAP pertenecen a la Capa 7 (Aplicación) del modelo OSI. Esta es la capa más alta, donde los datos tienen significado semántico para las aplicaciones. JSON/SOAP son formatos de representación de datos que las aplicaciones entienden y procesan.

- ¿Qué beneficios tiene utilizar un formato como JSON/SOAP?

Legibilidad humana: Puedes leer y debuggear fácilmente los datos
Interoperabilidad: Cualquier lenguaje de programación puede parsear JSON/SOAP
Estructura clara: Los datos están organizados con claves y valores, evitando ambigüedad

- ¿Qué ventajas y desventajas tiene el enfoque Pub/Sub de Kafka?

Ventajas:

- Desacoplamiento: Producer y consumer son independientes, pueden fallar sin afectarse mutuamente
- Escalabilidad: Puedes agregar múltiples producers/consumers sin modificar el sistema

Desventajas:

- Complejidad adicional: Necesitas mantener el broker (Kafka) como componente extra
- Latencia: Hay un intermediario, no es comunicación directa.
- ¿Para qué aplicaciones tiene sentido usar Kafka? ¿Para cuáles no?

Tiene sentido para: Sistemas IoT con miles de sensores enviando telemetría o Streaming de datos en tiempo real (logs, métricas, eventos)
NO tiene sentido para: Aplicaciones simples con pocos usuarios (overhead innecesario) y/o Comunicación request-response tradicional (mejor usar REST/HTTP)

3.4.2 IOT EN ENTORNOS CON RESTRICCIONES

- ¿Qué complejidades introduce el tener un payload restringido (pequeño)?

Se introdujen las siguientes complejidades:

  - Pérdida de precición
  - Necesidad de diseño de protocolos binarios
  - Mayor complejidad del código

- ¿Cómo podemos hacer que el valor de temperatura quepa en 14 bits?

Un valor en 14 bits se representa como 2^{14} - 1, estos datos deben ser escalados, se redondean al entero más cercano y de ahí se empaqueta usando operaciones bitwise.

- ¿Qué sucedería si ahora la humedad también es tipo float con un decimal?¿Qué decisiones tendríamos que tomar en ese caso?

De pasar a ser un flotante este dato ya no cabría en 1 o 2 bytes sin perder precisión. En este caso se podría multiplciar el valor por 10 y luego se escala el dato para usar 1 byte (se pierde precisión).

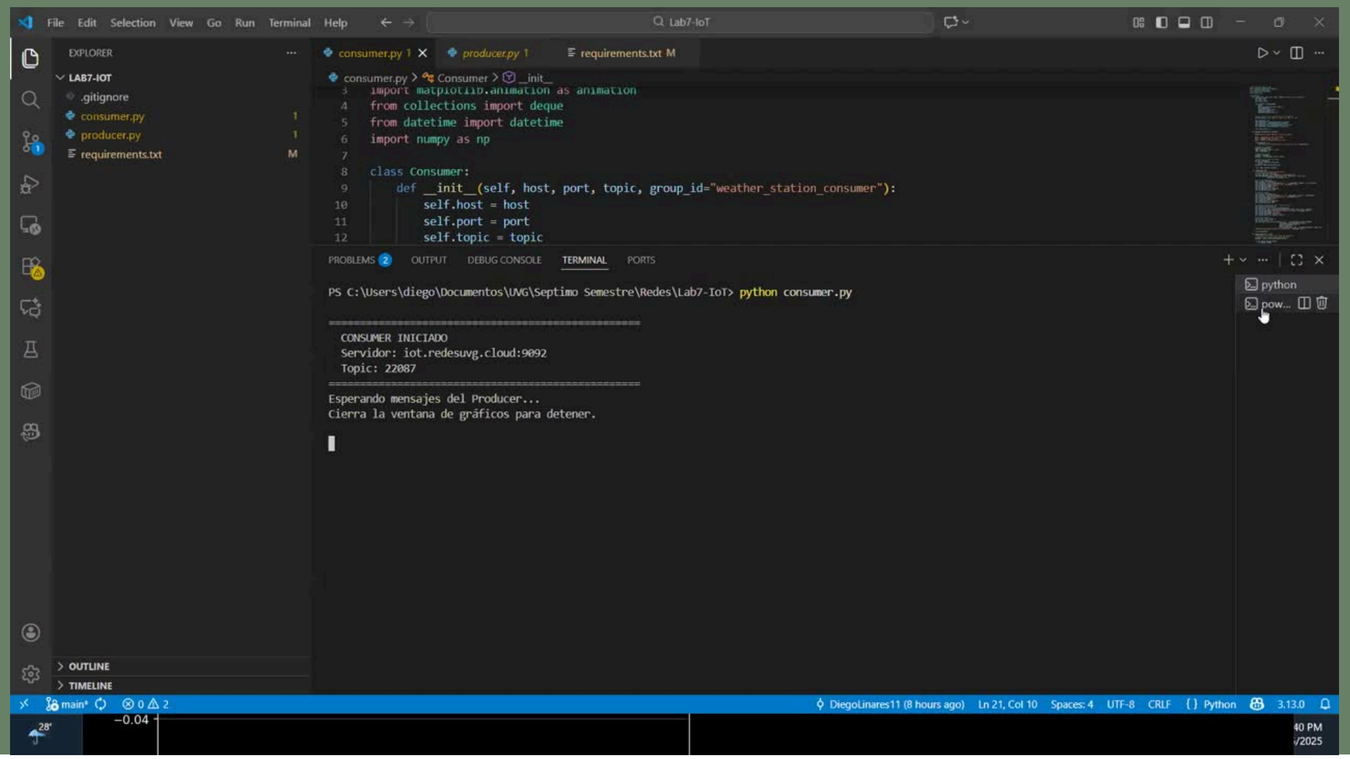
- ¿Qué parámetros o herramientas de Kafka podrían ayudarnos si las restricciones fueran aún más fuertes?

Kafka Message Headers

Apache Avro / Protobuf / MessagePack

Kafka Stream Processing

VISUALIZACION GRAFICAS



OBJETIVOS

Objetivo 1:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse elit libero, egestas vitae bibendum a, gravida non quam.

Objetivo 2:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse elit libero, egestas vitae bibendum a, gravida non quam.

CONCLUSIONES

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse elit libero, egestas vitae bibendum a, gravida non quam. In mattis, mauris ac mollis feugiat, sapien mi cursus tellus, non lacinia lorem ex a ante.

MUCHAS GRACIAS

TRABAJO GRUPAL | CLASE DE 1ºB

ALBA CASTRO, ALBERTO NAVARRO Y CRISTINA GALLEGO